Guidance on Installation and Configuration of a Zabbix Monitoring System for CDF Offline

Gerald Guglielmo

(CD-doc-2659)

Table of Contents

1 Introduction

An important feature of any operational system is a set of metrics for assessing the system's performance. Metrics can be integrated into the system or developed outside the system through aggregation of system information. A key component of production systems is often monitoring of the operational aspects of the system in real or near real time.

There are many commercial systems as well as freely available systems available on the web, and Zabbix is one of the freely available systems. Zabbix is an open source monitoring system that can be downloaded at http://www.zabbix.com/. The systems consists of agents that can be run on the systems to be monitored, a server that collects data from the agents and stores it in a database, and a web interface for monitoring and configuration based on php.

Installing and commissioning of a monitoring system for CDF Offline would hopefully improve the efficiency of operations and in the longer term reduce the overall effort load required to maintain these systems. This document is designed to provide guidance on how Zabbix could be installed and commissioned for CDF Offline. However some parts are sufficiently general and thus it could be used as a guide for other projects.

2 Specification of Hardware Requirements

The Zabbix system can place a heavy load on the operating system. A quick look using top shows that the zabbix_agent and the zabbix server place only a small load on the system for a small configuration of one host and 42 parameters at the default rate. However, on a single CPU 3Ghz processor it is showing a load average around 7. This load is mainly from the Postgres postmaster, therefore for even from a quick minimal test one is going to need good hardware to run the system. The *Zabbix User's Manual* has some guidelines.

2.1 Basic Hardware Recommendations

To monitor all of the nodes in the current CAF system the *Zabbix User's Manual* recommendation is for at least 2GB of Memory and a dual core CPU. Disk space depends on the amount of data per monitor cycle and how long it will be saved. However the smallest system recommendation for 50 hosts is much less performant than the initial desktop system tested, and yet the desktop was seeing a moderate load. This indicates that guidance in the manual may be on the optimistic side and it might be better to go with a more robust system. If an intermediate deployment becomes available, more accurate estimates and understanding are likely to be generated. *What did CMS use Paul?*

2.2 Estimating Database Size

The *Zabbix User's Manual* has formulas for calculating the history, trend, and event (when a trigger fires) data. There is also a small amount of configuration data on order of 10MB. The formulas can be used to develop a way of quickly estimating the amount of data we might need to store. The basic formulas are:

history=days*(items-per-host*hosts/period)*24*3600*50 bytes

trend=days*(items-per-host*hosts/1800)*24*3600*128 bytes

event=days*(events/period)*24*3600*130 bytes

The hardest of these to estimate is the event data as it is how often triggers fire in the system. A very conservative events/period value of 1 per second can be used for a well operating system even with a large number of items monitored. Events should be rarer than this, as this represents problems occurring on average once per second in the production system. Scale down event data based on experience.

The above formulas can be converted into scaling factors by using reasonable base value assumptions. For instance, it is handy to see the formulas above reduced to bytes/item-per-host/minute/30-days for history and trend data. In other words, what is the impact of storing one item every minute from one host for 30 days? If we later want to know the value for 10 items on each of 1000 hosts we just need to multiplty by 10*1000. To scale to 60 days would be an additional factor of two. Let a month represent 30 days.

history=2.06MB/item/host/minute/month

trend=0.20MB/item/host/minute/month

Assume 20 items per host and 1000 hosts, then we have

history=40GB/minute/month

trend=3.4GB/minute/month

If we further increased the trend saving period to 3 months, that would boost the trend to about 12GB, and the history plus trend to about 52 GB.

The overly pessimistic event data for one month would be

event=321MB

and if keep for a year, 3.8GB. That would mean 56GB for the scenario. Scaling for CDF means just scaling the history and trend data. *Jerry try to get some estimates here.*

3 Configuration of the OS Environment

A list of software requirements for Zabbix are defined in the *Zabbix User's Manual* which is available online at http://www.zabbix.com/. On Scientific Linux Fermi 4.2, the installed version of php, version 4.3.9, has a bug that cannot properly parse static function declarations. There are different ways around this issue and this section will discussion some of the options.

The Apache web server also needs to be running.

3.1 SLF 4.X Environment

3.1.1 Additional RPMs for SLF 4.2

This is a listing of additional rpms that may need to be installed for SLF 4.2. To get a late enough version of the curl libraries, it was necessary to get a newer source rpm from elsewhere and rebuild it with rpmbuild (rpmbuild --rebuild curl-7.15.5-2.el5.src.rpm). Once rebuilt from sources, the installation was easy.

Rebuilt from source rpms:

- curl-7.15.5-2.i386.rpm
- curl-devel-7.15.5-2.i386.rpm
- curl-debuginfo-7.15.5-2.i386.rpm

3.1.2 Running Under PHP 4.3.9

Any rpms that need to be installed to get the web interface working? Support provided by gd libraries did not come by default on my machine. Jason or Paul?

The php 4.3.9 version has a parsing bug that cannot handle static function declarations. Once can work around this problem by removing the keyword “static” from the declaration of all functions in the php web interface scripts. In version 1.4.4 of Zabbix it appears that the only script with static function declarations is php/include/copt.lib.php.

3.1.3 Running Under PHP 5.2.5

For the build of php version 5.2.5, it was necessary to have httpd-devel installed. The version of httpd-devel in the SLF4.2 yum repository is sufficient, but may not be installed by default (yum install httpd-devel*).

Installed via yum:

- httpd-devel-2.0.52-38.sl4.2.i386.rpm

Built and installed from tar files:

- gd-2.0.33.tar.gz
- php-5.2.5.tar.gz

The gd libraries were need so php could support the web interface. Building these libraries was simple.

```
./configure
make
make install
```

The configuration and building of php shown will include support for postgres, presumably for mysql the configuration step is similar with a swap of the database engine name.

```
./configure --enable-bcmath --with-apxs2=/usr/sbin/apxs --enable-gd-native-ttf --with-pgsql --with-config-file-path=/etc –with-gd=/usr/local
make
make install
cp php.ini-recommended /etc/php.ini
```

It is necessary to set the timezone in the php.ini file to avoid annoying warnings. It is also necessary to change the maximum execution time to 300 seconds or the Zabbix checks on initial installation will fail. Edit php.ini and add/modify the following lines

- date.timezone= America/Chicago
- max_execution_time=300

In order for Apache to use the newly build php, Apache needs to be configured to load the module. The configuration for Apache on SLF 4.2 machines lists the php module in two places, which is an error, but for consistency and to these instructions detail changes in both places. In /etc/httpd/conf/httpd.conf, change from php4 to php5:

- LoadModule php5_module   /usr/lib/httpd/modules/libphp5.so

and in /etc/httpd/conf.d/php.conf (duplicate-but way php4 configured), change from php4 to php5:

- LoadModule php5_module modules/libphp5.so

3.2 SLF 5.X Environment

SLF 5.X comes with php 5.1.6, it is believed that this version of php does not have a problem parsing static function declarations based on articles posted on the Internet discussing Zabbix installation on RHEL 5. Therefore a standard SLF 5.X installation, perhaps with a yum install “php*”, should be compatible with the Zabbix distribution as is. It is highly likely however that the timezone setting and max... will still need to be set in php.ini, especially since the latter setting is a requirement of Zabbix and not a general value for the setting.

Hans Wenzel provided some specific information based on http://www.muck.net/?p=16, on additional packages needed for SLF 5.1. First install all the necessary packages as root:

yum -y install ntp php php-bcmath php-gd php-mysql httpd mysql gcc mysql-server mysql-devel net-snmp net-snmp-utils net-snmp-devel net-snmp-libs curl-devel

Also create the user account for zabbix:

useradd zabbix

3.2.1 Running Under PHP 5.1.6

It is necessary to set the timezone in the php.ini file to avoid annoying warnings. It is also necessary to change the maximum execution time to 300 seconds or the Zabbix checks on initial installation will fail. Edit php.ini and add/modify the following lines

- date.timezone= America/Chicago
- max_execution_time=300

Note that the former setting is required in all version 5 releases of php and the latter is required in all versions of php by Zabbix.

4 Configuration of the Backend Database

Among the supported database engines Zabbix supports, Postgres and Mysql are of the most interest at the laboratory. Either one should work and the instructions in the *Zabbix User's Manual* do a good job of covering the steps needed once a basic database has been configured and is running. Because the web server hosting the Zabbix web interface will generally run on the same host as the database, the database need only listen on the loopback interface 127.0.0.1.

4.1 Postgres Database Configuration

Hans Wenzel has installed postgres on an SLF 5.1 machine and this section

is based on his experience with that installation. as user postgres: Install postgres and make sure it is running. (check tail -f /var/lib/pgsql/pgstartup.log if there are any problems starting up the database server).
/usr/bin/createuser zabbix -createdb --no-createrole --pwprompt
/usr/bin/createuser zabbix --createdb --no-createrole --pwprompt
createdb -O zabbix zabbix

Should include here information on default Postgres configuration.

The *Zabbix User's Manual* should be used for the details of this process. Assuming a user account has been created by the initial configuration of the database engine, there are only a few steps that need to be accomplished. First log into the database and create a new database called zabbix. After logging out the remaining tasks are to create the database tables, load initial data and load initial images. Zabbix provides sql scripts for accomplishing each of these tasks.

4.2 Mysql Database Configuration

Should include here information on default Mysql configuration.

The *Zabbix User's Manual* should be used for the details of this process. Assuming a user account has been created by the initial configuration of the database engine, there are only a few steps that need to be accomplished. First log into the database and create a new database called zabbix. After logging out the remaining tasks are to create the database tables, load initial data and load initial images. Zabbix provides sql scripts for accomplishing each of these tasks.

5 Building and Installation of Zabbix

The Zabbix User's Manual is fairly good on how to build the server, agents and front end web interface. The web interface is php code and does not need compilation. However it is important to do a “make clean” before the configure step to avoid problems due to any previous attempts at make that might have occurred.

The following steps show how to build and install the server, agents and front end web interface using Postgres. For Mysql one would just replace –with-postgres with –with-mysql. The build can be done as any normal user, however the installation step should be done as root. Note that the web interface code will need to be copied to a new directory called zabbix under the root directory specified in the Apache configuration for web pages (by default it seems to be /var/www/html).

```
make clean
./configure --with-pgsql --enable-server --enable-agent --with-libcurl
make
make install
mkdir /var/www/html/zabbix
cd frontends/php
cp -R * /var/www/html/zabbix
```

6 Zabbix Agents and Senders

Collection of data on the remote systems is for the most part done by Zabbix agent daemons. The agents have a predefined set of keys, or checks, they understand, some which take parameters, and one can also add user defined keys in a configuration file. In many cases this mechanism is sufficient to do all of the data collection on the remote systems. However in the event a long running process needs to periodically make data available for monitoring, there is a sender command that can be used to send the data directly to the server. This section will not discuss the sender command, zabbix_sender, and instead will focus on the agent daemon, zabbix_agentd.

6.1 General Zabbix Agent Configuration

The agent configuration file contains parameter settings for communication between the agent and server, user defined keys which allow for personalized checks, and some general operating parameters. For the most part the default settings are sufficient. One will have to change at least the Server setting IP address as by default it uses the loopback interface 127.0.0.1. If one wants to enable active checks from the server side, then there is a Hostname parameter that needs to be changed, otherwise the defaults for both server and agent ports are fine. Any special checks that Zabbix does not defined by default will need to be added.

6.1.1 User Defined Keys

The *Zabbix User's Manual* does a fair job of explaining how to create user keys in the agent configuration file. The declaration starts with *UserParameter=*, followed by a key name, and then a comma followed by the command to execute. For example, a key name of f*irefly.ping* is defined to execute a ping command that is further processed using Unix pipes and commands.

```
UserParameter=firefly.ping,ping -c 1 firefly | grep mdev | cut -d'=' -f2 |
cut -d'/' -f1
```

As a test, one can check the newly defined key by invoking the agent with the test flag:

```
bash-3.00$ ./zabbix_agentd -t"firefly.ping" -c ./zabbix_agentd.conf
firefly.ping                              [t| 0.040]
```

The output has the same basic format as the predefined keys. It appears to be fairly easy to also set up user parameters that take flexible arguments:

```
UserParameter=firefly.grep[*],grep -c $1 $2
UserParameter=firefly.echo[*],echo "$1 to $2"
```

Notice that you need to specify how to use the arguments, but that is not surprising. Again testing the defined key will work is easy using the test flag to the agent.

```
bash-3.00$ ./zabbix_agentd -t"firefly.echo[soup, nuts]" -c
./zabbix_agentd.conf
firefly.echo[echo "soup to nuts"]           [t|soup to nuts]


bash-3.00$ ./zabbix_agentd -t"firefly.grep[the,
/home/gug/NovaQuestions.txt]" -c ./zabbix_agentd.conf
firefly.grep[grep -c the /home/gug/NovaQuestions.txt]  [t|24]
```

6.2 Pushing Agent Configuration to Hosts

Need a way to push the same basic configuration, although if active checks are enabled there would need to be one line different in each one, to all of the monitored nodes. *Jason and Paul?*

7 Zabbix Server

Configuration of the Zabbix Server requires setting the interface the server will listen on, which by default is the loopback interface 127.0.0.1. The default port should be sufficient. It will also be necessary to define the database user name in the configuration file, /etc/zabbix/zabbix_server.conf, and the password for that user if needed. Note how to deal with the password is coupled to how the database was originally configured, *Jason and Paul?*

8 Starting Zabbix Agents and Servers

Ideally there should be a mechanism for starting and stopping the server and agents in an automated way to avoid having to do this by hand on multiple machines. This could be done through the init.d mechanism, or any other mechanism in use for the systems. *Jason and Paul?*

Hans Wenzel has setup a single host environment under SLF 5.1 to start the sever daemon when the operating system is restarted. First edit /etc/init.d/zabbix_agentd and /etc/init.d/zabbix_server and change the BASEDIR definition to be:

BASEDIR=/usr/local/zabbix

8.1 Starting Zabbix Server Daemon

Jason and Paul?

For automatically starting the server daemon first prepare the server daemon start script for use by chkconfig. Edit /etc/init.d/zabbix_server (note the # Hash marks are required), add near the top, just below *#!/bin/sh*:
# chkconfig: 345 95 95
# description: Zabbix Server

Next using chkconfig, configure the automatic starting and stopping of server daemon.
chkconfig -level 345 zabbix_server on

8.2 Starting Zabbix Agents on Remote Hosts

Jason and Paul?

For automatically starting the agent daemon first prepare the agentd start script, as was done for the server daemon, for use by chkconfig. In /etc/init.d/zabbix_agentd (Note the # hash marks are once again necessary), add near the top, just below *#!/bin/sh:*
# chkconfig: 345 95 95
# description: Zabbix Agentd
Next using chkconfig, configure the automatic starting and stopping of the agentd daemon.
chkconfig -level 345 zabbix_agentd on

9 Web Configuration of Zabbix

On the first attempt to login as the super administrator, user Admin, the web interface will take you through a series of configuration steps. Part of this process is to check for required functionality in php. After the initial configuration one will then need to declare to the system what hosts to monitor, checks to perform, triggers to evaluate, users to allow and permissions to grant, graphs to provide, etc.

9.1 Initial Web Configuration

There are a series of steps the web interface will walk you through the first time you log in under the Admin account (super administrator). One of the final tasks is to manually save and then copy a php configuration file into the zabbix area.

zabbix.conf.php /var/www/html/zabbix/conf

9.2 Zabbix Templates

Templates work as containers to hold items to monitor and triggers associated with those items so that they can be applied consistently to multiple hosts. Associating a template with a host essentially defines all of the items and triggers in the template for the new host at one time. This greatly simplifies management of monitoring similar hosts. For example, all Linux servers have a /tmp directory and we might want to monitor space free for /tmp and trigger if it falls below 10%. Defining a check on /tmp and a trigger for less than 10% on that check in a template means all hosts associated with that template will have the check and trigger defined. Templates can be viewed as grouping hosts that need the same monitoring and triggering together.

The creating and configuring of templates is done in separate steps. From the configuration page for hosts, one toggles to “Templates” and then selects “Create Template.” Once created one needs to add items and triggers. The forms for creating items and triggers can be accessed via the select link on the line listing the template on the “Templates” page, or through the items or triggers page and selection the desired template in the pull down menu.

Adding items is fairly straight forward. One needs to define a name for the item and the key, which essentially looks like the input to the Zabbix Agent when testing a parameter. For example, a key defined as “*firefly.echo[soup, nuts]*” assuming the agents understand that definition.

Adding triggers are a little more difficult, but once the pattern is understood it becomes easier. Again a name has to be given for the trigger along with an expression to be evaluated. The easiest way to define an expression is to select insert and then choose the select button on the popup menu. A condition will need to be defined in the popup form, but again the form gives some guidance here.

9.3 Zabbix Host Groups

Whereas templates group together hosts that need the same monitoring and triggering, host groups work to group components of a larger system together. These components may require different monitoring checks, thus different templates, or they may not. The concept here is to bring together all of the various components that should be monitored as one larger unit. For example, consider a system of two logging nodes and one database node that comprise a logging system. The two logger nodes might have one template shared, while the database has a different one that has database checks defined. From a monitoring perspective, to see if the entire logging system is working one wants to monitor all three nodes. To accomplish this task one could create a Logger host group and associate all three nodes with that host group. This association would then allow someone to access monitoring for the entire system by just selecting the one host group.

9.4 Users and User Groups

Users are individual accounts for logging in and accessing the system. User groups represent a container for mapping users to hosts through host groups and defining access permissions on a granularity of host groups. In this way access to monitoring and configuration of monitoring for hosts is controlled by the Zabbix super administrator. Thus it is fairly easy to define access controls for users once you know how. It should be noted however this is not documented in the *Zabbix User's Manual* at this time.

9.4.1 Zabbix Users

Adding a new user can easily be done as the Zabbix super administrator, user Admin. From the administration page, click on Create User and fill in the requested information. Most of it is straight forward and it will tell you if you miss any required information when you try saving it. In a simple installation one might create an administration account for defining hosts, triggers, etc, and just use the guest account for monitoring. In a more complex environment one might want separate administrators and viewing accounts for separate functional systems monitored, for example experiments or services.

By default Zabbix defines a super administrator account, called Admin, and a guest account without a password. By default the guest account has no privileges and thus cannot do any monitoring or administration. Whether staying simple and using the guest account, or creating new accounts for monitoring, the enabling of permissions is done through user groups.

9.4.2 Zabbix User Groups

One can add a new user or just for simplicity start with the guest account which by default is unable to view anything. From the administration page, use the pull-down toggle button to switch to "User groups". At this point you can either create a new group, or click on one of the existing groups to bring up a form for configuring a user group. The main things you want to do with a user group are: add a list of users to it; add the host groups for read-write or read-only. By default everything is configured as deny.

To illustrate the process, assume a new user account has been created called CAFRead, and the goal is to create and configure a new user group. One could just update an existing group by selecting it from the “User Groups” page, but in this example a new user group will be created. Starting on the Administration page for “Users” toggled to "User Groups" and select “Create Group.” A form will appear that will allow one to create and configure the new user group. Note that the form is identical to the one used to update existing user groups, only in this case the fields are not yet set. For group name one would give "CAF Monitoring" and then under Users select add to add the users to the group. In this example CAFRead would be selected. All that remains now is to add permissions for any host groups to associate with the user group.

Of the three possible options for host groups, read-write, read-only, and deny, by default all host groups are defined as deny. There are separate lists and buttons to manage them for each category. Thus access permissions are defined on the host group level. In our example we just want read-only and are assuming the host groups “CAF Headnodes” and “CAF servers” were previously defined. Using the appropriate “add” button these host groups can be added to the read-only list. Once the configuration is saved, then user CAFRead will be able to view all hosts defined in those two host groups.

10 Operational Support

Support of the Zabbix installation covers hardware and operating system for the server, deployment of the Zabbix agents to remote nodes, and installation and operations of the Zabbix server and frontend in a production environment. Hardware and operating system support could be provided by different functional units within the organization if needed.

10.1 Hardware Support

There are different levels of hardware support that can be provided. Complete hardware support means handling all hardware issues down to replacement of bad components. However there is no requirement that the primary hardware support organization internally provide complete hardware support, and can delegate replacement activities to another organization. The proposal for the CDF CAF deployment is that the Fermilab Experiment Facilities department the provide the same level of support that they currently provide for other CAF hardware. *Jason is current support 9x5?*

10.2 Operating System Support

Support of the basic operating system includes installation, upgrades, and diagnostic support of the server and web interface machine. The operating system should be a standard operating system that the supporting organization is comfortable supporting and is compatible with the Zabbix requirements. It is believed that Zabbix will work well under SLF 4.X or SLF 5.X, and either of these should be sufficient. As with the hardware support, the proposal for CDF CAF is that the Fermilab Experiment Facilities department the provide support.

10.3 Zabbix Operational Support

Support will be needed for installing, configuring, upgrading and maintaining the Zabbix software. The main purpose of Zabbix is for system monitoring, which includes operating system level and higher level services monitoring. While support for Zabbix could be handled by either the service providers or the operating system support personnel, in this case it would be best if the operating system support personnel take responsibility. The service providers should not have configuration control, but should provide configuration requests and allow the Zabbix support team to make the changes. By restricting the access to configuration changes the reliability of the Zabbix system can be maximized. It is also likely the support team members will better understand the load issues on the Zabbix system. The proposal for CDF CAF is that the Fermilab Experiment Facilities department the provide support.

11 Monitoring Support Assignments

Assignments for monitoring and responding to triggers and alerts in the Zabbix system will need to be worked out. In general operating system issues should be assigned to the group providing operating system support. However it is important that access to the overviews and charts be more open so CAF support personnel can have read access too. This allows the CAF support to learn to spot trends and report them perhaps even before a trigger is activated.

In addition to operating system checks, like mount points, it is expected that some service monitoring will be available. It may be desirable to send alerts directly to the CAF team in these instances.

11.1 Configuring Actions

Ideally the first time a trigger fires an action should be sent out. However it would be nice to avoid a flood of messages coming from the trigger firing each time a check is made. Zabbix supports hysteresis with respect to firing and clearing triggers, which is a nice feature. In practical terms one could set up a trigger so that it fires when a level reaches 90%, but does not clear until that level falls below 70% for example. While that can be useful in preventing an edge effect from toggling on and off, it is not necessary in preventing actions due to a continued state of on or off. Another customization that can be of use is to send a differently formatted message if the trigger turns on compared to when it turns off.

These customizations are possible in Zabbix. Sending email when a trigger first transitions to true, and a separate one for when if first transitions to false, without a barrage of emails during the ON state is fairly straight forward once the pattern is known. For example consider a simple case for a check on the existence of a file. The trigger in this example is called “Alert file found” and equates to TRUE if the item check for the file is successful. To allow customization of the message subject and body, and even the recipient, to this specific trigger one condition is defined as "Trigger description like" and value “Alert.” The “Trigger severity” setting in this example isn't necessary, but shown as a possible option for conditions. The “Trigger value” shows the state of the trigger after evaluation. In the first definition the “Trigger value” causes the action to be performed when the state transitions to true, and the second the setting is for the transition to false. In this way a separate action is defined for both transitions.

<u>Action to send on TRUE</u>
Trigger description like "Alert"
Trigger severity >= "High"
Trigger value = "TRUE"

<u>Action to send on FALSE</u>
Trigger description like "Alert"
Trigger severity >= "High"
Trigger value = "FALSE"

12 Developing Batch Host Declarations

The Zabbix web interface does not currently support declaring multiple hosts with a similar configuration in one pass. Instead the interface in Zabbix version 1.4.4 allows defining one host at a time. Therefore to declare a large number of hosts one must invoke the same sequence of steps repeatedly. In principle one could write a set of php scripts to perform this activity in a loop and thus add many hosts at one time. This section provides a proof of principle on this idea. Initially the tests were done at the command line using the command line php parser. However, in a production environment one would most likely want to perform this through the web interface which will better enforce access controls. Since the code is php scripts, doing this just means placing them in the same place as the Zabbix php (/var/www/html/zabbix on my test setup) scripts and knowing the proper URL path (http://firefly.fnal.gov/zabbix/zabbixTest4.php).

While code will be provided here, it is not considered production quality and only serves as a guide on how to perform the key steps.

12.1 Creating Batch Host Declaring Functionality

In order to add a bunch of hosts at one time, one needs to define the data that represents the configuration of these hosts. Formatting of this data can be done in different ways and so can reading the data by the processing scripts. A nice way of doing the data is to format it as php code and import the data file through the equivalent of an include statement (in this case a require_once line). This gets around having to write a parser as the php parser can now do the work.

The proof of principle here is for a simple case without profile information. With some investigation however, it should be possible to enhance the functionality to provide profile information also. That may not be a trivial amount of work if any of those fields are not strings, but it can systematically be investigated and implemented.

One of the more difficult aspects of getting this code working was dealing with templates and groups. Neither of these turned out to be string objects, neither are handled or structured in the same way, and both depend upon querying the database for know instances. In both cases the object are arrays, but indexing for groups starts at zero, while indexing for templates uses keys pulled from the database (10026 for example). Furthermore, to get usable results from the higher level query methods one has to have proper privileges. Still it can be done.

12.2 Batch Host Declaring Code Example

In this proof of principle there are three scripts. The first is the data script, the second is the main processing script, and the third is a set of helper functions.

12.2.1 Data Input In PHP Formatting

```
<?php
$status=HOST_STATUS_MONITORED; // need to check
$useip=1;
$port=10050; //need to check
```

```
// list of existing groups
//$group_names=array();
$group_names=array("Linux servers",
                   "CAF servers");
// newgroup, "" if none
$newgroup="CAF Headnodes";
// list of templates
//$template_names=array();
$template_names=array("Template_Linux",
                      "Template_CAF",
                      "Template_Dummy");
// list of hosts
$hosts=array("icaf01.fnal.gov",
             "icaf02.fnal.gov",
             "icaf03.fnal.gov",
              "icaf04.fnal.gov",
             "iserver99.fnal.gov");
// list of ip addresses
$ips=array("192.168.1.1",
           "192.168.1.2",
           "192.168.1.3",
           "192.168.1.4",
           "192.168.17.6");

$dns=array();
$count=count($hosts);
// fill dns array with "" as we are using IP addresses
for($i=0; $i<$count; $i++) {
  $dns[] = "";
 }

?>
```

12.2.2 Main Processing Script

```
<?php
echo "Fourth test using require_once mechanism for data import\n";
#ini_set("include_path", ".:./zabbix-1.4.4/frontends:./zabbix-
1.4.4/frontends/php:./zabbix-1.4.4/frontends/include");

// variable to flag config.inc.php to not call setup.inc.php because I
// do not have a configured zabbix installation. comment out maybe for
// working installation (gug_cmdline would need to be unset)? Matches
hack to
// config.inc.php at the moment.
$gug_cmdline=1;
// requires a later php than is on slf4
require_once "include/config.inc.php";
require_once "include/hosts.inc.php";

// just include the data here
require_once ("./zabbixTest4.inc.php");
require_once ("./zabbixMyFunc.inc.php");

// this is just a dump of the data read reformatted
echo "  Groups=(";
foreach ($groups as $gname) {
  echo "$gname,";
}
echo ")\n";
echo "  Templates=(";
foreach ($templates as $tname) {
  echo "$tname,";
}
echo ")\n";
echo "  Newgroup=$newgroup\n";
echo "  Hosts=(";
foreach ($hosts as $hname) {
  echo "$hname,";
}
echo ")\n";
echo "  IPs=(";
foreach ($ips as $iname) {
  echo "$iname,";
}
echo ")\n";
// map group format to database entries
$gmap=groups_cmdline_map();
$groups=array();
foreach ($gmap as $id => $name) {
  if (in_array($name,$group_names)) {
     echo "Matched Group $id $name\n";
     $groups[]=$id;
  }
}

// map template format to database entries
$tmap=templates_cmdline_map();
//var_export($tmap);
$templates=array();
foreach ($tmap as $id => $name) {
  if (in_array($name,$template_names)) {
     echo "Matched Template $id $name\n";
     $templates[$id]=$name;
  }
}
var_export($groups);
var_export($templates);
// now try actually adding hosts
```

```
$index=0;
foreach ($hosts as $host) {
  echo "$index $ip[$index]\n";
  echo "trying
add_host($host,$port,$status,$useip,$dns[$index],$ips[$index],$templat
es,$newgroup,$groups);";
    add_host($host,$port,$status,$useip,$dns[$index],$ips[$index],$temp
lates,$newgroup,$groups);
            $index=$index+1;
}
?>
```

12.2.3 Helper Functions Script

```
<?php
//echo "Function testing for templates and groups\n";
// requires a later php than is on slf4
require_once "include/config.inc.php";
require_once "include/hosts.inc.php";

function get_accessible_hosts(&$user_data) {
    $perm_res = PERM_RES_STRING_LINE;
    $perm           = PERM_READ_ONLY;
    $resdata = '$host_data["hostid"]';
    $userid =& $user_data['userid'];
    $user_type =& $user_data['type'];

    $where = '';
    $sql = 'select distinct n.nodeid,n.name as node_name,h.hostid,h.host,
min(r.permission) as permission,ug.userid '.
           ' from hosts h left join hosts_groups hg on hg.hostid=h.hostid '.
           ' left join groups g on g.groupid=hg.groupid '.
           ' left join rights r on r.id=g.groupid and
r.type='.RESOURCE_TYPE_GROUP.
           ' left join users_groups ug on ug.usrgrpid=r.groupid and
```

```
ug.userid='.$userid.
        ' left join nodes n on '.DBid2nodeid('h.hostid').'=n.nodeid '.
                  $where.' group by
h.hostid,n.nodeid,n.name,h.host,ug.userid '.
        ' order by n.name,n.nodeid, h.host, permission desc, userid desc';


   $db_hosts = DBselect($sql);
   $processed = array();
   while($host_data = DBfetch($db_hosts))
   {
      if(is_null($host_data['nodeid'])) $host_data['nodeid'] =
id2nodeid($host_data['hostid']);


      /* if no rights defined used node rights */
      if( (is_null($host_data['permission']) ||
is_null($host_data['userid'])) )
      {
         if( isset($processed[$host_data['hostid']]) )
            continue;


         if(!isset($nodes))
         {
            $nodes = get_accessible_nodes_by_user($user_data,
            PERM_DENY,PERM_MODE_GE,PERM_RES_DATA_ARRAY);
         }
         if( !isset($nodes[$host_data['nodeid']]) ||
$user_type==USER_TYPE_ZABBIX_USER )
             $host_data['permission'] = PERM_DENY;
         else
             $host_data['permission'] =
$nodes[$host_data['nodeid']]['permission'];
      }


      $processed[$host_data['hostid']] = true;
```

```
            $result[$host_data['hostid']] = eval('return '.$resdata.';');

        }

        unset($processed, $host_data, $db_hosts);

        if($perm_res == PERM_RES_STRING_LINE)
        {
            if(count($result) == 0)
                $result = '-1';
            else
                $result = implode(',',$result);
        }

        return $result;

    }

    function        get_accessible_groups($user_data)
    {
        global $ZBX_LOCALNODEID;
        $perm_mode = PERM_MODE_GE;
        $perm_res = PERM_RES_STRING_LINE;
        $perm = PERM_READ_LIST;
        $nodeid = null;
        $result = array();

        $userid =& $user_data['userid'];
        $user_type =& $user_data['type'];
        $resdata = '$group_data["groupid"]';

        $where = '';
```

```
   /* if no rights defined used node rights */
   $db_groups = DBselect('select n.nodeid as nodeid,n.name as
node_name,hg.groupid,hg.name,min(r.permission) as
permission,g.userid'.
                ' from groups hg left join rights r on r.id=hg.groupid and
r.type='.RESOURCE_TYPE_GROUP.
                ' left join users_groups g on r.groupid=g.usrgrpid and
g.userid='.$userid.
                ' left join nodes n on '.DBid2nodeid('hg.groupid').'=n.nodeid '.
                $where.' group by n.nodeid, n.name, hg.groupid, hg.name,
g.userid, g.userid '.
                ' order by n.name, hg.name, permission desc');


   $processed = array();
   while($group_data = DBfetch($db_groups))
   {
       if(is_null($group_data['nodeid'])) $group_data['nodeid'] =
id2nodeid($group_data['groupid']);
       /* deny if no rights defined */
       if( is_null($group_data['permission']) ||
is_null($group_data['userid']) )
       {
           if(isset($processed[$group_data['groupid']]))
               continue;


           if(!isset($nodes))
           {
               $nodes = get_accessible_nodes_by_user($user_data,
                       PERM_DENY,PERM_MODE_GE,PERM_RES_DATA_ARRA
Y);
           }


           if( !isset($nodes[$group_data['nodeid']]) ||
$user_type==USER_TYPE_ZABBIX_USER )
               $group_data['permission'] = PERM_DENY;
```

```
          else
              $group_data['permission'] =
$nodes[$group_data['nodeid']]['permission'];
        }
//      $processed[$group_data['permission']] = true;
        $processed[$group_data['groupid']] = true;

        $result[$group_data['groupid']] = eval('return '.$resdata.';');
    }
    unset($processed, $group_data, $db_groups);

    if($perm_res == PERM_RES_STRING_LINE)
    {
        if(count($result) == 0)
            $result = '-1';
        else
            $result = implode(',',$result);
    }
    return $result;
}



function groups_cmdline_map() {
    global $USER_DETAILS;
    $db_group=array();
    if ($USER_DETAILS['alias'] == 'Admin') {
        echo "debug: user Admin in groups_cmdline_map\n";
        $db_groups=DBselect("select distinct groupid,name from groups ".
            " where groupid in (".
            get_accessible_groups_by_user($USER_DETAILS,PERM_READ_
ONLY,null,null,get_current_nodeid()).
            ") order by name");
    } else {
        $db_groups=DBselect("select distinct groupid,name from groups ".
            " where groupid in (".
            get_accessible_groups($USER_DETAILS).
            ") order by name");
    }
    $group_map=array();
    while($db_group=DBfetch($db_groups))
    {
       $group_map[$db_group["groupid"]]=$db_group["name"];
    }
    return $group_map;
}


//groups_cmdline_map();
function templates_cmdline_map() {
    global $USER_DETAILS;
    if ($USER_DETAILS['alias'] == 'Admin') {
        echo "debug: user Admin in templates_cmdline_map\n";
        $accessible_hosts       =
get_accessible_hosts_by_user($USER_DETAILS,PERM_READ_ONLY);
    } else {
        echo "debug: user Guest in templates_cmdline_map\n";
        $accessible_hosts       = get_accessible_hosts($USER_DETAILS);
    }
    $sql = "select distinct h.* from hosts h where ";

    $sql .= DBin_node('h.hostid', $nodeid).
           " and h.hostid in (".$accessible_hosts.") ".
           " and h.status=".HOST_STATUS_TEMPLATE.
           " order by h.host,h.hostid";

    $db_hosts = DBselect($sql);
    $template_map=array();
    while($host = DBfetch($db_hosts))
    {
          $template_map[$host["hostid"]]=$host["host"];
    }
    return $template_map;
}


//$xxx=templates_cmdline_map();
//var_export($xxx);
//$yyy=groups_cmdline_map();
//var_export($yyy);
?>
```